LSTM System Identification

Data Preparation

In [ ]:

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dense, Dropout
import pandas as pd
from matplotlib import pyplot as plt
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
import seaborn as sns
#from datetime import datetime

#Read the csv file
df = pd.read_csv("upsample_min.csv")
df = pd.read_csv("upsample.csv")
df2=df.drop(df.columns[0], axis=1)
data = df['GLIR'].to_numpy()

split = 0.75

x1 = df['GLIR'].to_numpy()[:int(split*len(data))]
x1 = x1.reshape(len(x1),1)
y1 = df['Qo'].to_numpy()[:int(split*len(data))]
y1 = y1.reshape(len(y1),1)

x2 = df['GLIR'].to_numpy()[int(split*len(data)):]
y2 = df['Qo'].to_numpy()[int(split*len(data)):]

print(f"ukuran x train: {np.shape(x1)} ukuran y train: {np.shape(y1)}")
print(f"ukuran x test: {np.shape(x2)} ukuran y test: {np.shape(y2)}")

df2.head()
```

```
ukuran x train: (2772, 1) ukuran y train: (2772, 1)
ukuran x test: (924,) ukuran y test: (924,)
```

Out[ ]:

	GLIR	Qo
0	54.200000	165.000000
1	46.999293	162.692956
2	40.483007	160.443006
3	34.681550	158.260778
4	29.617764	156.156846

Train Data

Preprocessing Data

In [ ]:

```
#New dataframe with only training data
df_for_training_x = x1
df_for_training_y = y1

#LSTM uses sigmoid and tanh that are sensitive to magnitude so values need to be normalized
# normalize the dataset
scaler = StandardScaler()
scaler = scaler.fit(df_for_training_x)
scaler2 = scaler.fit(df_for_training_y)
df_for_training_scaled_x = scaler.transform(df_for_training_x)
df_for_training_scaled_y = scaler2.transform(df_for_training_y)

#As required for LSTM networks, we require to reshape an input data into n_samples x timesteps x n_features.
#In this example, the n_features is 5. We will make timesteps = 14 (past days data used for training).

#Empty lists to be populated using formatted training data
trainX = []
trainY = []

n_future = 1   # Number of days we want to look into the future based on the past days.
n_past = 14  # Number of past days we want to use to predict the future.

#Reformat input data into a shape: (n_samples x timesteps x n_features)
#In my example, my df_for_training_scaled has a shape (12823, 5)
#12823 refers to the number of data points and 5 refers to the columns (multi-variables).
for i in range(n_past, len(df_for_training_scaled_x) - n_future +1):
```

```
    trainX.append(df_for_training_scaled_x[i - n_past:i, 0:df_for_training_x.shape[1]])

for i in range(n_past, len(df_for_training_scaled_y) - n_future +1):
    trainY.append(df_for_training_scaled_y[i + n_future - 1:i + n_future, 0])

trainX, trainY = np.array(trainX), np.array(trainY)

print('trainX shape == {}.'.format(trainX.shape))
print('trainY shape == {}.'.format(trainY.shape))
```

```
trainX shape == (2758, 14, 1).
trainY shape == (2758, 1).
```

RNN LSTM Architecture & Training

In [ ]:

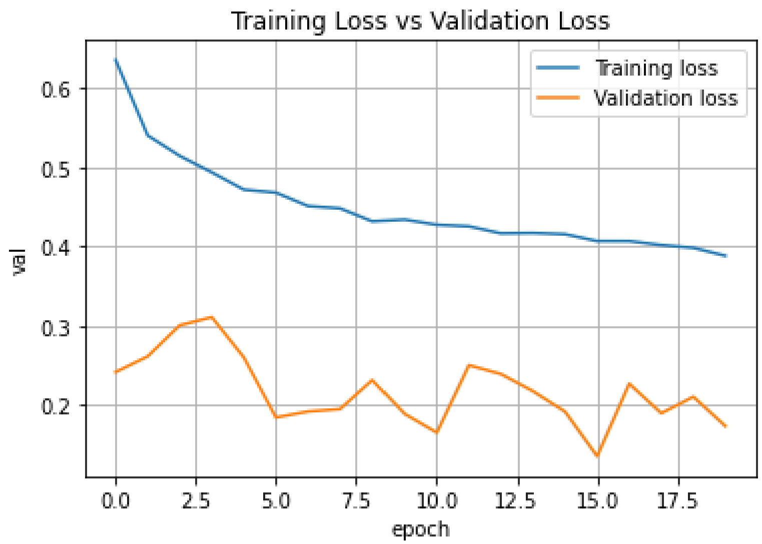
```
# define the Autoencoder model

model = Sequential()
model.add(LSTM(64, activation='relu', input_shape=(trainX.shape[1], trainX.shape[2]), return_sequences=True))
model.add(LSTM(32, activation='relu', return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(trainY.shape[1]))

model.compile(optimizer='adam', loss='mse')
model.summary("")


# fit the model
history = model.fit(trainX, trainY, epochs=20, batch_size=15, validation_split=0.1, verbose=1)

#model.save("RNN_model")
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_12 (LSTM)              (None, 14, 64)            16896


_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_12 (LSTM)              (None, 14, 64)            16896

 lstm_13 (LSTM)              (None, 32)                12416

 dropout_6 (Dropout)         (None, 32)                0

 dense_6 (Dense)             (None, 1)                 33

=================================================================
Total params: 29,345
Trainable params: 29,345
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
166/166 [==============================] - 6s 21ms/step - loss: 0.6360 - val_loss: 0.2415
Epoch 2/20
166/166 [==============================] - 3s 16ms/step - loss: 0.5405 - val_loss: 0.2613
Epoch 3/20
166/166 [==============================] - 3s 17ms/step - loss: 0.5145 - val_loss: 0.3006
Epoch 4/20
166/166 [==============================] - 2s 15ms/step - loss: 0.4937 - val_loss: 0.3109
Epoch 5/20
166/166 [==============================] - 3s 18ms/step - loss: 0.4718 - val_loss: 0.2600
Epoch 6/20
166/166 [==============================] - 3s 18ms/step - loss: 0.4680 - val_loss: 0.1840
Epoch 7/20
166/166 [==============================] - 3s 20ms/step - loss: 0.4511 - val_loss: 0.1915
Epoch 8/20
166/166 [==============================] - 3s 18ms/step - loss: 0.4481 - val_loss: 0.1945
Epoch 9/20
166/166 [==============================] - 3s 20ms/step - loss: 0.4317 - val_loss: 0.2312
Epoch 10/20
166/166 [==============================] - 3s 19ms/step - loss: 0.4339 - val_loss: 0.1884
Epoch 11/20
```

```
166/166 [==============================] - 3s 19ms/step - loss: 0.4273 - val_loss: 0.1648
Epoch 12/20
166/166 [==============================] - 3s 19ms/step - loss: 0.4254 - val_loss: 0.2501
Epoch 13/20
166/166 [==============================] - 3s 18ms/step - loss: 0.4165 - val_loss: 0.2389
Epoch 14/20
166/166 [==============================] - 3s 16ms/step - loss: 0.4169 - val_loss: 0.2172
Epoch 15/20
166/166 [==============================] - 3s 19ms/step - loss: 0.4154 - val_loss: 0.1913
Epoch 16/20
166/166 [==============================] - 3s 16ms/step - loss: 0.4067 - val_loss: 0.1348
Epoch 17/20
166/166 [==============================] - 3s 19ms/step - loss: 0.4067 - val_loss: 0.2267
Epoch 18/20
166/166 [==============================] - 3s 17ms/step - loss: 0.4017 - val_loss: 0.1893
Epoch 19/20
166/166 [==============================] - 3s 19ms/step - loss: 0.3982 - val_loss: 0.2102
Epoch 20/20
166/166 [==============================] - 3s 15ms/step - loss: 0.3881 - val_loss: 0.1730
```

In [ ]:

```python
xx = np.arange(0,len(history.history['loss']))

"""print(history.history['loss'])
print(history.history['val_loss'])
print(xx)"""

plt.figure(1)
plt.plot(xx,history.history['loss'], label='Training loss')
plt.plot(xx,history.history['val_loss'], label='Validation loss')
plt.title("Training Loss vs Validation Loss")
plt.xlabel("epoch")
plt.ylabel("val")
plt.legend()
plt.grid()
```

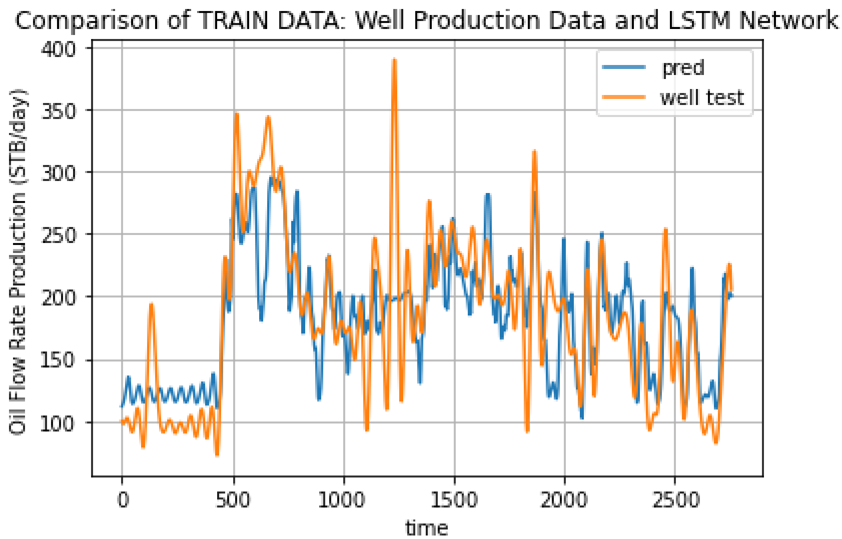


Predicting Values

In [ ]:
```
#Make prediction
#model = keras.models.load_model("RNN_Model")

n_days_for_prediction = 20
prediction = model.predict(trainX[:]) #shape = (n, 1) where n is the n_days_for_prediction

#Perform inverse transformation to rescale back to original range

prediction_copies = np.repeat(prediction, df_for_training_y.shape[1], axis=-1)
y_pred_future = scaler.inverse_transform(prediction_copies)

print('nilai pred:',y_pred_future)

yy = scaler.inverse_transform(trainY)

x_axis = np.arange(0,y_pred_future.shape[0])

print(f"ukuran y: {np.shape(yy)} ukuran y pred: {np.shape(y_pred_future)}")

plt.figure(2)
plt.plot(x_axis,y_pred_future, label='pred')
plt.plot(x_axis,yy, label='well test')
plt.title("Comparison of TRAIN DATA: Well Production Data and LSTM Network")
```

```
plt.xlabel("time")
plt.ylabel("Oil Flow Rate Production (STB/day)")
plt.legend()
plt.grid()
plt.show()
```

```
87/87 [==============================] - 1s 6ms/step
nilai pred: [[111.88711]
 [112.27278]
 [112.63561]
 ...
 [200.02582]
 [200.1059 ]
 [200.63802]]
ukuran y: (2758, 1) ukuran y pred: (2758, 1)
```



Metric

In [ ]:

```
import math
from sklearn.metrics import r2_score

MSE = np.square(np.subtract(yy,y_pred_future)).mean()

RMSE = math.sqrt(MSE)
print("Root Mean Square Error:")
print(RMSE)
```

```
r2 = r2_score(yy,y_pred_future)
print("\nR2 Value:")
print(r2)
```

```
Root Mean Square Error:
38.13771190701517

R2 Value:
0.6511904554187627
```

Forecasting Value/Test Data

In [ ]:

```
#New dataframe with only testing data
x2 = x2.reshape(len(x2),1)
y2 = y2.reshape(len(y2),1)
df_for_testing_x = x2
df_for_testing_y = y2

#LSTM uses sigmoid and tanh that are sensitive to magnitude so values need to be normalized
# normalize the dataset
scaler = StandardScaler()
scaler = scaler.fit(df_for_testing_x)
scaler2 = scaler.fit(df_for_testing_y)

df_for_testing_scaled_x = scaler.transform(df_for_testing_x)
df_for_testing_scaled_y = scaler2.transform(df_for_testing_y)

#As required for LSTM networks, we require to reshape an input data into n_samples x timesteps x n_features.
#In this example, the n_features is 5. We will make timesteps = 14 (past days data used for testing).

#Empty lists to be populated using formatted testing data
testX = []
testY = []

n_future = 1   # Number of days we want to look into the future based on the past days.
n_past = 14  # Number of past days we want to use to predict the future.

#Reformat input data into a shape: (n_samples x timesteps x n_features)
#In my example, my df_for_testing_scaled has a shape (12823, 5)
#12823 refers to the number of data points and 5 refers to the columns (multi-variables).
for i in range(n_past, len(df_for_testing_scaled_x) - n_future +1):
    testX.append(df_for_testing_scaled_x[i - n_past:i, 0:df_for_testing_x.shape[1]])
```

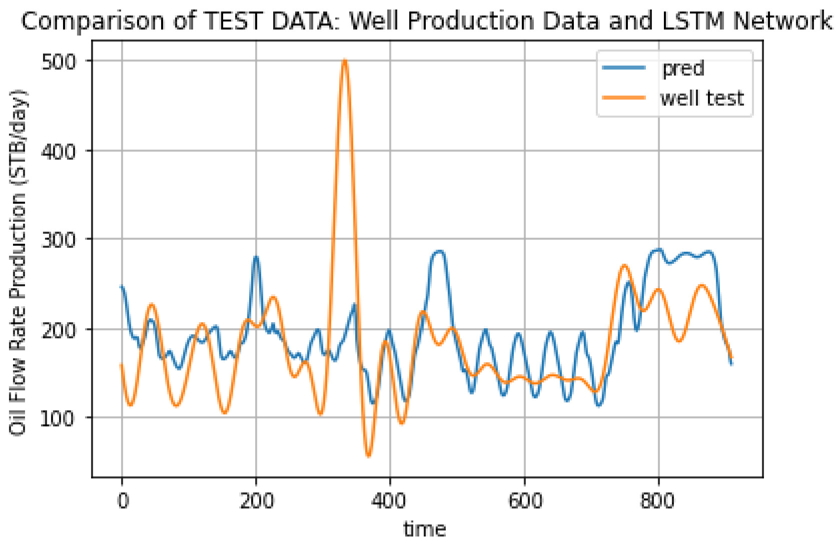
```python
for i in range(n_past, len(df_for_testing_scaled_y) - n_future +1):
    testY.append(df_for_testing_scaled_y[i + n_future - 1:i + n_future, 0])

testX, testY = np.array(testX), np.array(testY)

print('testX shape == {}.'.format(testX.shape))
print('testY shape == {}.'.format(testY.shape))
```

```
testX shape == (910, 14, 1).
testY shape == (910, 1).
```

In [ ]:

```python
#Make forecast
#model = keras.models.load_model("RNN_Model")

n_days_for_forecast = 20
forecast = model.predict(testX[:]) #shape = (n, 1) where n is the n_days_for_forecast

#Perform inverse transformation to rescale back to original range

forecast_copies = np.repeat(forecast, df_for_testing_y.shape[1], axis=-1)
y_fore_future = scaler.inverse_transform(forecast_copies)

#print('nilai pred:',y_fore_future)

yyy = scaler.inverse_transform(testY)

x_axis = np.arange(0,y_fore_future.shape[0])

print(f"ukuran y: {np.shape(yyy)} ukuran y pred: {np.shape(y_fore_future)}")

plt.figure(2)
plt.plot(x_axis,y_fore_future, label='pred')
plt.plot(x_axis,yyy, label='well test')
plt.title("Comparison of TEST DATA: Well Production Data and LSTM Network")
plt.xlabel("time")
plt.ylabel("Oil Flow Rate Production (STB/day)")
plt.legend()
plt.grid()
plt.show()
```

```
29/29 [==============================] - 0s 7ms/step
ukuran y: (910, 1) ukuran y pred: (910, 1)
```



Metric

In [ ]:
```
import math
from sklearn.metrics import r2_score

MSE = np.square(np.subtract(yyy,y_fore_future)).mean()

RMSE = math.sqrt(MSE)
print("Root Mean Square Error:")
print(RMSE)

r2 = r2_score(yyy,y_fore_future)
print("\nR2 Value:")
print(r2)
```

```
Root Mean Square Error:
62.53082207858559

R2 Value:
0.08141813482511318
```